Towards a triple mode common operator FFT for Software Radio systems

Ali Al Ghouwayel, Amin Haj-Ali, Zouhair El-Bazzal, Yves Louët

Towards a Triple Mode Common Operator FFT for SoftWare Radio Systems

Ali Chamas Al Ghouwayel, Amin Haj-Ali and Zouhair El-Bazzal
Lebanese International University
P.O.Box:146404 Mazraa, Beirut, LEBANON
Email:{ali.ghouwayel, amin.hajali, zouhair.bazzal}@liu.edu.lb

Yves Louët
SUPELEC-IETR, CS 47601
35576 CESSON-SEVIGNE Cedex, FRANCE
Email: yves.louet@supelec.fr

***Abstract*—A scenario to design a Triple Mode FFT is addressed. Based on a Dual Mode FFT structure, we present a methodology to reach a triple mode FFT operator (TMFFT) able to operate over three different fields: complex number domain $\mathbb{C}$, Galois Fields $GF(F_t)$ and $GF(2^m)$. We propose a reconfigurable Triple mode Multiplier that constitutes the core of the Butterfly-based FFT. A scalable and flexible unit for the polynomial reduction needed in the $GF(2^m)$ multiplication is also proposed. An FPGA implementation of the proposed multiplier is given and the measures show a gain of 18 % in terms of performance-to-cost ratio compared to a "Velcro" approach where two self-contained operators are implemented separately.**

I. Introduction

In recent years there has been an enormous proliferation of standards in broadcast television, radio and in mobile communications. Current examples include digital television (DVB, ISDB), digital radio (DAB), wireless LAN(Hiperlan, 802.11a, 802.11b, ..., 802.16m, ...), 2.5/3G, 4G and future mobile communications. These standards form the basis for an ever-growing number of sophisticated consumer electronic devices, each with the potential to sell in very high volumes.

In typical designs, these complex standards are implemented using dedicated architectures, which are optimized to reduce cost to the absolute minimum. Products developed using dedicated architectures are often difficult to upgrade in order to support changes to the standards or to add new features.

At the beginning of the 90's, a concept called *SoftWare Radio* (SWR) has emerged from demonstrations in military research to become a cornerstone of the third generation strategy for affordable, ubiquitous and global communications [1]. This SWR technology is a way to design a sufficiently programmable and reconfigurable architecture able to support many different transmission standards on a common platform. The reconfigurability of a SWR system can offer a range of benefits at different levels. A radio system implemented on a reconfigurable architecture can be upgraded to fix bugs or to add functionalities, and it can also support new standards as it is assumed that there is sufficient flexibility in the architecture.

The communication chains of different standards, intended to be implemented on a common platform, have some common signal processing operations such as channel coding, modulation, equalization, etc. In order to exploit to a great advantage the commonalities among these communication tasks for different standards, one need firstly to identify these commonalities and secondly find the optimal way to implement a generic hardware platform with programmable modules. In this sense, a technique called *parametrization* has been introduced [2]. The key idea is to get an optimal sharing between hardware and software resources and find a best way to reuse some hardware and software modules without affecting the system's performances.

This paper addresses the problem of parametrization technique under the CO approach where it is in line with the optimal design of a SWR system intended to support several communication standards. In [8], we have investigated the frequency processing of cyclic codes particularly RS codes with the aim to insert the classical FFT operator, initially used for complex Fourier transform, in the encoding and decoding processes of RS codes. By examining the characteristics of the Fourier transform FFT-GF2 used to process the encoding and some decoding processes of classical RS codes defined over $GF(2^m)$, we found that the adequacy of its structure to the structure of FFT defined over $\mathbb{C}$ (FFT-$\mathbb{C}$) is challenged by the transform length. That is, the most efficient algorithms applied to a transform of length 2^m in the case of FFT-$\mathbb{C}$ computations cannot be applied to compute FFT-GF2 since its transform length is of the form $2^m - 1$. For this, we thought to seek out a transform satisfying the FFT-$\mathbb{C}$ criteria while keeping in mind to find a way to include the FFT-GF2, used in the RS coding defined over $GF(2^m)$, in the intended common and reconfigurable FFT structure.

To be able to exploit the whole FFT-$\mathbb{C}$ structure we explored finite field transforms having a highly composite length. The candidate transforms were the Fermat transforms defined over $GF(F_t)$ [3] [5]. These transforms are used to deal with a specific class of RS codes defined over $GF(F_t)$. These codes were studied in 1976 where the authors have emphasized the importance of their codeword length allowing the use of efficient algorithms to compute their associated Fermat transforms [4] [6]. These codes were also recommended for the use in spacecraft communications [7] where the RS(256,224) over $GF(257)$ was studied to be used together with a convolutional code.

As a first contribution, we have investigated the redesign of the FFT-$\mathbb{C}$ in such a way to be able to provide two functionalities: complex Fourier transform and Fermat transform. Conceptually, the design of such a dual mode operator implies

the design of arithmetical operators capable to operate over the two domains: C and $GF(F_t)$. We have proposed reconfigurable architectures for the multiplier, adder and subtracter [9] [10]. These operators implemented on the complex butterfly (available in the FFT-$\mathbb{C}$) led to a reconfigurable butterfly that will constitute the core of the dual mode operator. Based on the FFT-$\mathbb{C}$ structural strategy, we have designed the architecture of the DMFFT operator [11].

To evaluate the complexity and speed performances of this operator, we have considered its implementation on ALTERA's FPGA devices. Compared to a Velcro FFT/FNT operator, the DMFFT presented an important gain in terms of ALUTs and memory saving. We have shown that for a transform length $N = 64$ implemented with different wordlengths ($9 \leq n_c \leq 16$), the DMFFT presents a memory saving between 20 and 30%, a gain in ALUTs and performance-to-cost ratio gain that go from 9.2 % up to 26 % and from 9.7 % up to 37.4 % respectively. This deviation of gain in ALUTs and in performance-to-cost ratio is directly related to the wordlength, so as n_c increases, these gains decrease. For $N = 256$, the DMFFT presents the same memory saving and the other gains evolve in the same manner of that of DMFFT-64 but with lower values. This is due to the fact that the complexity of the DMFFT architecture is mainly dominated by the FFT-$\mathbb{C}$ architecture complexity which increases with the increase of the wordlength and the transform length. The high n_c values are necessary to get a good FFT-$\mathbb{C}$ computations precision while in the Velcro FFT/FNT, a 9-bit wordlength is sufficient to implement the FNT. Then, for a transform length $N \leq 256$, the use of the DMFFT is largely efficient and allows to process RS codes which have codeword length $N \leq 256$ and whose principals are N=64, 128 and 256.

In [12], we have proposed a hardware design and efficient implementation of the FFT over $GF(2^m)$. The proposed design is based on the cyclotomic decomposition of polynomials representing the input data. A very high throughput rate is attained, that makes the FFT to be a Common Operator while it is capable to produce its functionality at least twice faster than the existing solutions.

Nevertheless, RS codes implemented in the various actual standards (ADSL, ATSC, DVB, GSM,...) are codes defined over $GF(2^m)$. Then, the inclusion of the finite field Fourier transform defined over $GF(2^m)$ as an extra functionality to be provided by our designed CO is a necessity to be on actual standardization's trend.

In this paper we propose a scenario that aims to combine the FFT-GF2 with the DMFFT by exploiting the same logical cells at the gate level to obtain a reconfigurable Triple Mode FFT (TMFFT) operator. We propose a generic combined multiplier able to perform a standard binary multiplication and $GF(2^m)$ multiplications for m=6, 7 and 8. We also show that this multiplier can be easily implemented into the reconfigurable multiplier proposed in [10] which leads to a triple mode multiplier able to operate over three different fields: $\mathbb{C}$, $GF(F_t)$ and $GF(2^m)$. Having this triple mode multiplier and taking into account that the addition in $GF(2^m)$ can be realized by XOR gates, the necessary arithmetical tools to implement the FFT-GF2 are available in the DMFFT operator.

II. Towards a Combined Triple Mode FFT Operator (TMFFT)

We propose a scenario that aims to combine the FFT-GF2 structure with the DMFFT structure on the same die area to obtain a combined and reconfigurable operator TMFFT. This combination can be achieved if the following two steps can be realized.

1) Providing the $GF(2^m)$ operations, mainly the multiplications, with the binary multipliers implemented in the DMFFT operator.
2) Incorporation of the FFT-GF2 physical structure into the DMFFT structure.

We begin the discussion with the first step where the approach for a combined multiplier exploits the fact that the partial product generations of both $GF(2^m)$ multiplier and standard binary multiplier can be performed using the same array and interconnections between cells.

A. Basic binary multiplication

Let us consider the basic principle of the standard binary arithmetic multiplier. In 1964, Wallace [13] introduced a notion of a carry-save tree constructed from one-bit full adders as a way of reducing the number of partial product bits in a fast and efficient way. Later, Dadda [14] refined Wallace's method by defining a cell placement strategy that minimizes the number of full-adders and half adders, at the cost of a larger carry propagate adder. For our multiplier design, we will consider the Wallace's method since it is structurally more regular.

Wallace'method is based on parallel counters and the multiplication of two binary numbers is performed in the following sequence.

- Form all partial products in parallel with an array of AND gates.
- Reduce the partial products through a series of reduction stages to two numbers by strategically applying (3,2) and (2,2) counters. Architectures of counter (3,2) (or full adder) denoted by "W_3" and (2,2) counter (or half adder) denoted by "W_2" are illustrated in Fig. 1.
- Sum the two numbers produced in step 2 using a fast carry-propagate adder to generate the final product.

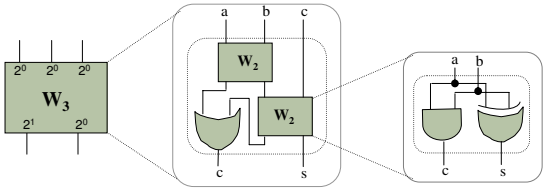

Fig. 1. The W3 and W2 architectures

B. $GF(2^m)$ multiplication

As for the $GF(2^m)$ multiplication, it is viewed as a polynomial multiplication modulo $f(x)$, where $f(x)$ is the irreducible polynomial characterizing the field $GF(2^m)$.

Let us consider the multiplication of two $GF(2^m)$ numbers A and B. A and B can be represented by means of the vector basis $\{\alpha^{m-1}, ..., \alpha^1, \alpha^0\}$, where α is a primitive element of the field, as

$$A = a_{m-1}\alpha^{m-1} + ... + a_1\alpha^1 + a_0\alpha^{m-1},$$

$$B = b_{m-1}\alpha^{m-1} + ... + b_1\alpha^1 + b_0\alpha^{m-1},$$

where a_i, b_i are coefficients in $GF(2)$.

The polynomial multiplication $C(x) = A(x)B(x)$ mod $f(x)$ can be calculated by firstly building the partial products

$$P_i(x) = A(x)b_i, \qquad for \quad i = 0, ..., m-1.$$

Secondly, since the polynomial's coefficients belong to $GF(2)$, all partial products $P_i(x)$ have to be added modulo 2. A partial result $C_p(x)$ can be obtained with

$$C_p(x) = \sum Pi(x) \ mod \ 2$$

$$C_p(x) = c_0 + c_1x + ... + c_{2m-2}x^{2m-2},$$

which is obviously not an element of the field $GF(2^m)$ and should be reduced modulo $f(x)$.

A $GF(2^m)$ multiplier then performs two basic operations. The product of two elements and the modulo $f(x)$ correction. The first operation can be performed by ANDing the corresponding a_i and b_i, for $i = 0, ..., m-1$, and subsequently adding the partial products modulo 2. These partial products must be arranged in rows, with each row shifted i positions to the left as shown in Fig. 2. This step is similar to the standard binary product with the only difference that the sum of partial products in this case is done modulo 2. Thus, an opportunity to exploit cells and the interconnection structure of partial product generator of a typical binary multiplier unit is possible. The modulo correction should be performed separately.

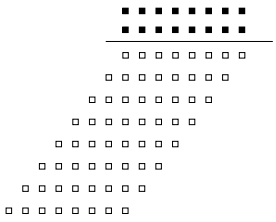

Fig. 2. Partial product matrix

Early designs of $GF(2^m)$ multipliers used a serial approach. Although serial $GF(2^m)$ multipliers have low hardware requirements, they are very slow. Consequently, several parallel designs have been proposed in the literature [15] [16] [17] [18]. In [19] [20], the idea to combine $GF(2^m)$ with a standard binary multiplier on DSP processor was investigated. In [19], the proposed design is based on a Wallace tree multiplier which has been modified to perform either conventional binary or $GF(2^m)$ multiplication. Their polynomial reduction (or modulo correction) introduces a linear delay. In [20], a new wiring scheme, to avoid adding carries of partial product reduction, is proposed and a parallel polynomial reduction is used. The authors designed a multiplier capable of performing either 16-bit two's complement or unsigned multiplication, or two independent 8-bit $GF(2^8)$ multiplications. In the following, we propose a general approach allowing the implementation of any $GF(2^m)$ multiplier, for $6 \leq m \leq 8$, in the reconfigurable multiplier proposed in [10]. We start here from the standard binary structure of the multiplier shown in Fig. 3. The partial product generation or bitwise ANDing of a_ib_i is

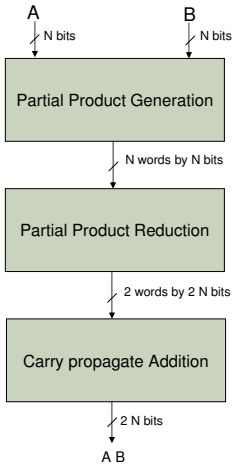

Fig. 3. Steps for N by N multiplication

performed regardless of the type of multiplication ($GF(2^m)$ or standard binary multiplication). The partial product reduction should be redesigned in such a way to avoid the carries propagation if the $GF(2^m)$ multiplication is to be performed. This can be realized by reconfiguring the wire connects of the "W3" cells. The new wiring scheme is illustrated in Fig. 4. This wiring scheme can be easily realized in a reconfigurable way on FPGA devices using a reprogrammable LUTs. Arriving at

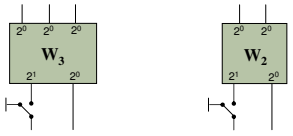

Fig. 4. The (3,2) and (2,2) counters in $GF(2^m)$ multiplication

the last step (the polynomial reduction), it can be performed in parallel by using the method considered in [17]. Let $P(\alpha)$ be the extended result before the polynomial reduction. $P(\alpha)$ can be expressed as

$$P(\alpha) = \sum_{i=m}^{2m-2} p_i\alpha^i + \sum_{i=0}^{m-1} p_i\alpha^i, \qquad (1)$$

where α^i for $m \leq i \leq 2m-2$ can be substituted by

$$A_i(\alpha) = \sum_{j=0}^{m-1} a_{i,j}\alpha^j.$$

The $A_i(\alpha)$ are the canonical representations in the field's base of the field elements α^i that appear in the first summation in Equation 1.

Let us consider an example of multiplication of two elements A and B in $GF(2^4)$. Let $f(x) = x^4 + x + 1$ be the primitive polynomial and

$$A = \alpha^{10} = 0.\alpha^3 + \alpha^2 + \alpha + 1 = (0111),$$
$$B = \alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1 = (1111).$$

The multiplication operation is described in Fig. 5. The classical method of multiplication shown in the upper left side of Fig. 5 is expressed by hardware circuit which is based on Wallace tree. This tree, originally designed to perform the standard binary multiplication, is modified in such a way to avoid the carries propagation between the cells constituting the tree. In this approach we have considered the disconnection of the carry's outputs and the corresponding neighboring cell inputs that are set to zero. The disconnected wires are represented by black dots. This wiring scheme can be realized by means of reprogrammable LUT1s that connect the carry outputs of each A^iB^i unit to the corresponding entries of the $W3$ cells. A LUT1 maintain the carry propagation (by connecting the input to the output) in the case of standard binary multiplication and set its output to zero in the case of $GF(2^m)$ multiplication.

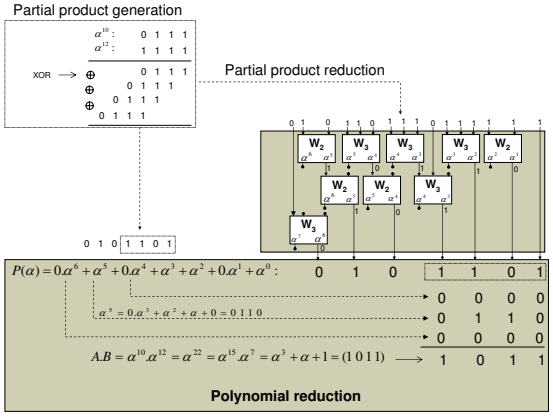

Fig. 5. Multiplication of two elements in $GF(2^4)$

Fig. 5 shows an example of $GF(2^4)$ multiplier whose whole architecture, except the polynomial reduction, is implemented on the standard binary multiplier. This implementation can be extended for any $GF(2^m)$ multiplier provided that the size of the original binary multiplier can support the size of the $GF(2^m)$ multiplier to be implemented.

In the next section, we propose a reconfigurable architecture of a combined multiplier that can support either a binary multiplication or $GF(2^m)$ multiplication for m=6, 7 and 8.

III. Proposed Triple Mode Multiplier

In this section we consider the combination of $GF(2^m)$ multiplier, for $m = 6, 7, 8$, with the standard binary multiplier. Fig. 6 shows a block diagram of the combined triple mode multiplier 8×8 bits. In this design, we restrict the sizes of GF multipliers according to the code lengths of RS codes used in the practical applications.

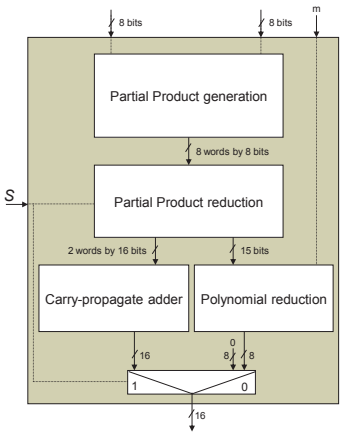

Fig. 6. Block diagram of the combined multiplier

The architecture we propose is based on the Wallace's tree for the partial product reduction. A tree of size 8×8 is sufficient to perform $GF(2^8)$ multiplication. However, the tree size is fixed according to the desired precision for the complex Fourier transform. As discussed in [11], a 13 bit-wordlength represents a good complexity-precision tradeoff. Thus, for $GF(2^m)$ multiplication, the Wallace's tree can process any two words for $m \leq 13$ but a further attention should be taken into account for the realization of the polynomial reduction. For this, in the following we restrict the values of m to the more practical ones. The multiplier receives a control signal S to pilot the switching from the standard binary addition to modulo 2 addition and to manage the configuration of the interconnection between the neighboring cells according to the model previously shown in Fig. 5. The multiplier performs the partial product reduction regardless of the size of the operands. If the size is smaller than 13, the operands can be concatenated by "0". The rest of the architecture is the carry-propagate adder (for the standard binary multiplication) and the polynomial reduction unit (for the $GF(2^m)$ multiplication).

The part which will be influenced by the variable length of the GF multiplier is the polynomial reduction unit since the reduction is based on the binary representation of the powers of α which varies with the order of the Galois field. For this, we propose a reconfigurable polynomial reduction unit as shown in Fig. 7.

This polynomial reduction unit receives 15 bits from the partial product reduction block and a parameter m which defines the size of $GF(2^m)$ multiplication to be performed. According to the chosen GF, the corresponding powers of α are selected from the ROM blocks. By ANDing these powers

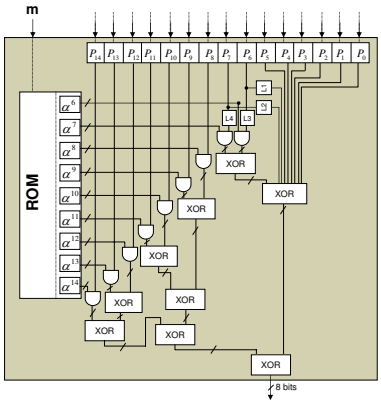

Fig. 7. Parallel polynomial reduction for $m = 6$, 7 and 8.

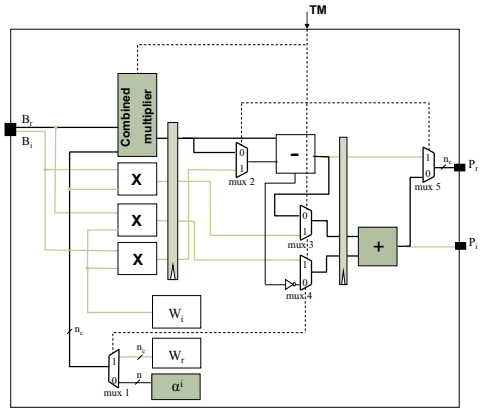

Fig. 8. Triple mode multiplier

TABLE I
CONFIGURATION OF THE LUT1S

multiplier size	L1	L2	L3	L4
$m = 6$	0	0	P_6	P_7
$m = 7$	P_6	0	0	P_7
$m = 8$	P_6	P_7	0	0

of α with the corresponding P_i, for $m \leq i \leq 2m-2$, and XORing their results with the word $\{P_{m-1} \ldots P_1 \ P_0\}$, the $GF(2^m)$ multiplication is provided on 8 bits. Four LUT1s ($L1$, $L2$, $L3$, $L4$) are needed to select the corresponding entries of the two first XOR gates depending on the $GF(2^m)$. These LUT1s are configured to provide their outputs according to Table I. If $m < 8$, the most significant bits of P_is are pre-initialized to zero and the rest of bits will contain the useful result. The latency of the polynomial reduction unit is equivalent to the time needed to perform one 8-bit AND and four 8-bit XOR.

The combined multiplier shown in Fig. 6 can be easily integrated within the reconfigurable multiplier designed in [10] leading to triple mode multiplier (Fig. 8) able to perform three different multiplications: (i) conventional binary multiplication (ii) multiplication over $GF(F_t)$ and (iii) multiplication over $GF(2^m)$. We note that n_c and n represent the complex and Galois field wordlengths respectively.

With this multiplier and by taking into account that a modulo 2 addition can be realized by means of XOR gates, the arithmetical resources required to design the TMFFT are designed.

To achieve the TMFFT scenario, further studies, considered as future, will focus on the design of a triple mode butterfly, appropriate dataflow schedule and a new wiring scheme allowing the integration of the structure of the $GF(2^m)$-FFT within the DMFFT operator.

IV. FPGA IMPLEMENTATION OF THE PROPOSED TRIPLE MODE MULTIPLIER

In order to evaluate the complexity of the proposed multiplier circuit, we consider its implementation on FPGA (Field Programmable Gate Arrays) devices. The complexity C of the FPGA-based circuit is determined by the number of logic blocks called Adaptive LUTs (i.e. ALUTs). The performance is measured in terms of the execution time T expressed in nanoseconds (ns). The advantage the common operator approach presents is emphasized when compared to an obvious approach called $Velcro$ approach. This latest considers the separately implementation of two self-contained operators: Dual Mode Multiplier and $GF(2^m)$ multiplier.

The comparison of the two approaches can be drawn by evaluating an overall metric that takes into account both the hardware complexity and the execution time. This metric, called performance-to-cost ratio and denoted by η, is expressed as $\frac{1}{TC} * 10^6$ [21]. The higher the metric η, the better the complexity-performance tradeoff is.

The choice of the target device is based on the considered devices in [10] [11]. We have written a synthesizable VHDL code[1] of the dual, Velcro and triple mode multipliers. Table II summarizes the complexity and performances measures for different number of bits n that represents the multiplier size. As shown, the integration of the $GF(2^m)$ multiplier causes an excess of only 6% of the total complexity. The metrics η_V and η_C denotes the performance-to-cost ratio of the Velcro and Common Operator approach respectively. The last row of Table II indicates the performance-to-cost ratio gain which is around 18 % in favor of the common operator approach. These measures show a good result which enforces the effectiveness of our adopted approach of reconfigurability and common operators.

[1]All experiments described in this paper were performed on a PC (Intel(R) Core(TM)2 Duo CPU, 2.1 GHz, 3 GB of memory) running Windows Vista. The VHDL code was synthesized using Quartus II version 6 and implemented on STRATIX II, EP2S15F484C3 Device with the option "Standard Fit" as the level of the Fitter's effort.

TABLE II
IMPLEMENTATION RESULTS OF THE DUAL AND TRIPLE MODE MULTIPLIERS ON STRATIX II, EP2S15F484C3 DEVICE

Multiplier	** n=6	n=8	n=10
Dual Mode	103 ALUTs 3.97 ns	194 ALUTs 4.1 ns	289 ALUTs 4.86 ns
Velcro	131 ALUTs 3.97 ns	245 ALUTs 4.13 ns	364 ALUTs 4.87 ns
Triple Mode	109 ALUTs 3.97 ns	206 ALUTs 4.17 ns	307 ALUTs 4.9 ns
$\eta = \frac{1}{TC} * 10^6$	$\eta_V = 1922$ $\eta_C = 2310$	$\eta_V = 988$ $\eta_C = 1164$	$\eta_V = 564$ $\eta_C = 664$
Performance-to-cost ratio gain	**20.1** %	**17.8** %	**17.7** %

** n: nb. of bits

V. CONCLUSION

We have presented a hardware design and efficient implementation of a triple mode multiplier able to operate over three different fields. A parameterizable polynomial reduction unit is proposed allowing the execution of the $GF(2^m)$ multiplication for various sizes of Galois field. The proposed multiplier is implemented on FPGA devices and the obtained measures show an excess of only 6% in terms of ALUTs compared to the complexity of the dual mode multiplier proposed in [10]. In terms of performance-to-cost ratio, the common operator approach represented by the triple mode multiplier outperforms the Velcro approach by a gain of around 18 %. An approach to design a TMFFT operator is described which will be further developed in near future works. The intended TMFFT operator can be used in two different contexts: modulation/demodulation OFDM and Reed Solomon encoding and decoding over $GF(F_t)$ and $GF(2^m)$.

REFERENCES

[1] J. Mitola, *Software Radio Architecture*, Wiley, 2000.

[2] F. Jondral, A. Wiesler and R. Machauer, *A Software Defined Radio Structure for 2nd and 3rd Genaration Mobile Communications Standards*, IEEE 6th Int. Symp. On Spread-Spectrum Tech. and Appli., New Jersey, USA, Sept. 6-8. 2000.

[3] R. C. Agarwal and C. S. Burrus, *Fast digital convolution using Fermat transforms*, in Southwest IEEE Conf. Rec., Houston, Tex., pp. 538-543, Apr. 1973.

[4] J. Justesen, *On the Complexity of Decoding Reed-Solomon Codes*, IEEE Trans. Inform. Therory, vol. IT-22, pp. 237-238 Mar. 1976.

[5] R. C. Agarwal and C. S. Burrus, *Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering*, IEEE Trans. on Acou. Spee. and Sig. Proces., vol. ASSP-22, no. 2, Apr. 1974.

[6] I. S. Reed, T. K. Truong and L.R. Welch, *The Fast Decoding of Reed-Solomon Codes Using Fermat Transforms*, IEEE Trans. Inform. Theory, vol. IT-24, no. 4, July 1978.

[7] M. R. Best; H. F. A. Roefs, *Technical assistance channel coding investigation (spacecraft Telemetry)*, [Final Report] 1981.

[8] A. Al Ghouwayel, Y. Louët and J. Palicot, *A Reconfigurable Architecture for the FFT Operator in a Software Radio Context*, IEEE ISCAS'2006, Island of Kos, Greece, May 2006.

[9] A. Al Ghouwayel, Y. Louët and J. Palicot, *A Reconfigurable Butterfly Architecture for Fourier and Fermat Transforms*, IEEE WSR'2006, Karlsrhue, Germany, March 2006.

[10] A. Al Ghouwayel, Y. Louët and J. Palicot, *Complexity Evaluation of a Re-Configurable Butterfly with FPGA for Software Radio Systems*, IEEE PIMRC'07, Athens, Greece, September 2007.

[11] A. Al Ghouwayel and Yves Louet, *FPGA implementation of a reconfigurable FFT for multi-standard systems in software radio context*, IEEE Transactions on Consumer Electronics, Vol. 55, No. 2, May 2009.

[12] Ali Al Ghouwayel, Yves Louët, Amor Nafkha and Jacques Palicot, *On the FPGA Implementation of the Fourier Transform over Finite Fields* $GF(2^m)$, IEEE ISCIT'2007, Sydney, Australia, Oct 16-19, 2007.

[13] C. S. Wallace, *A Suggestion For A Fast Multiplier*, IEEETransactions on Computers, vol. EC13, pp. 14-17, Feb. 1964.

[14] L. Dadda, *Some Schemes for Parallel Multipliers*, Alta Frequenza, vol. 34, pp. 349-356, 1965.

[15] E. D. Mastrovito, *VLSI Designs for Multiplications over Finite Fields* $GF(2^m)$, in Proc. Sixth Int. Conf. Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes (AAECC-6), pp. 297-309, 1988.

[16] C. S. Yeh, I. S. Reed, and T. K. Troung, *Systolic Multipliers For Finite Fields GF(2m)*, IEEE Transactions on Computers, vol. c-33, pp. 357-360, Apr. 1984.

[17] L. Gao and K. K. Parhi, *Custom VLSI Design of Efficient Low Latency and Low Power Finite Field Multiplier for Reed-Solomon Codec*, In IEEE International Symposium on Circuits and Systems ISCAS'01, pp. IV 574-577, 2001.

[18] Texas Instruments, *TMS320C64x Technical Overview*.

[19] W. Drescher, G. Fettweis and K. Bachmann, *VLSI Architecture For Non-Sequential Inversion Over* $GF(2^m)$ *Using the Euclidean Algorithm*, in Int. Conf. On Signal Processing Applications and Technology, 1997

[20] J. Garcia and M. J. Schulte, *A Combined 16-bit Binary and Dual Galois Field Multiplier*, In IEEE International Symposium on Circuits and Systems ISCAS'02, pp.63-68, 2002.

[21] William W.H. Yu and Shanzhen Xing *Fixed-Point Multiplier Evaluation and Design with FPGA*, Proceedings of SPIE, vol. 3844, September 1999.